EPiC Series in Computing
Volume 108, 2025, Pages 169–189
Proceedings of 12th Int. Workshop on Applied Verification for Continuous and Hybrid Systems

ARCH-COMP 2025 Category Report: Falsification

Tanmay Khandait[1], Deyun Lyu[2]*, Paolo Arcaini[2], Georgios Fainekos[3], Federico Formica[4], Sauvik Gon[5], Abdelrahman Hekal[6], Atanu Kundu[5], Claudio Menghi[7,4], Giulia Pedrielli[1], Rajarshi Ray[5], Quinn Thibeault[1], Masaki Waga[8], and Zhenya Zhang[9,2]

[1] Arizona State University (ASU), Tempe, USA
{tkhandai,giulia.pedrielli,quinn.thibeault}@asu.edu
[2] National Institute of Informatics (NII), Tokyo, Japan {lyudeyun,arcaini}@nii.ac.jp
[3] Toyota Motor North America, Research & Development georgios.fainekos@toyota.com
[4] McMaster University, Hamilton, Canada {formicaf,menghic}@mcmaster.ca
[5] Indian Association for the Cultivation of Science, Kolkata, India
{ugsg2584,mcsak2346,rajarshi.ray}@iacs.res.in
[6] Newcastle University, Newcastle upon Tyne, UK a.waleed-elsayed-aly-hekal@newcastle.ac.uk
[7] University of Bergamo, Bergamo, Italy claudio.menghi@unibg.it
[8] Kyoto University, Japan mwaga@fos.kuis.kyoto-u.ac.jp
[9] Kyushu University, Japan zhang@ait.kyushu-u.ac.jp

Abstract

We report the results from the falsification category of the 2025 competition in the Applied Verification for Continuous and Hybrid Systems (ARCH) workshop. We summarize the rules for this year's competition, the experimental settings, and benchmark models. We provide background on the participating teams and tools. Finally, we present and discuss the results of the competition.

Data: https://gitlab.com/goranf/ARCH-COMP

1 Introduction

The Applied Verification for Continuous and Hybrid Systems (ARCH) competition (a.k.a. ARCH-COMP) compares state-of-the-art tools for testing and verifying hybrid systems. It is a yearly competition organized in different categories. This document refers to the *falsification category* and is related to the 2025 editions. Reports related to previous editions (2017–2024) are available online [9, 10, 15, 14, 13, 16, 30, 24]. A recent report [1] describes the overall format

*The first author led the validation effort for the falsification category. The second author proposed five new models for this year's competition. The remaining authors represent all participants who have contributed results and/or text to this report and they are listed alphabetically.

of the competition and its organization. It also documents the experiences from the current and past editions of the competition, reflections, and lessons learned.

This report refers to the falsification category of the competition. This category targets the analysis of executable models. The participants need to find inputs that falsify requirements expressed in temporal logic with time bounds, encoded in *Metric Temporal Logic (MTL)* [25] or *Signal Temporal Logic (STL)* [29]. Typical approaches to finding these inputs use simulation-based techniques. These techniques search for initial system configurations and time-varying inputs subject to given constraints. They typically employ quantitative metrics [17, 19] to measure how close a given input is to violating a requirement ("robustness semantics") and stop when a falsifying input is detected.

As part of the competition, we maintain a set of benchmark models that can be seen as a baseline for research in the area [12]: We encourage authors working in this domain to compare their solutions with the one presented in this report.

The 2025 competition was structured as in previous editions. Participants agreed on the benchmarks to be considered. Then, they ran the experiments on their machines and submitted the results, including concrete input traces that witnessed falsification. The first author of this paper performed the validation step. The validation step analyzes the input traces submitted by the participants to confirm whether they falsified the requirements. The second author proposed five new models for this year's competition (SB1–SB5). Compared with the previous edition [24] the changes are as follows:

- We had no new tools participating in this edition. Three tools (EXAMNET, Moonlight, OD) did not participate. The remaining tools ARIsTEO [31], ATheNA [18], `FalCAuN` [36], FlexiFal (formerly known as NNFal) [27], ForeSee [39], and Ψ-TaLiRo [35] confirmed their participation.
- We introduced a new benchmark named Synthetic Benchmark (SB), consisting of five models automatically generated using `FalBenchGen` [38]. These benchmarks aim to address the lack of falsification benchmarks and are included for the first time in this year's competition.
- We updated our validation scripts to support the PM model, which was not covered during the last edition of the competition, and the five new Synthetic Benchmark models.

We remark that this report does not aim to summarize the research results for this area. The interested reader can refer to recent survey articles [7, 8].

This report is structured as follows. Section 2 introduces our benchmark models and requirements. Section 3 summarizes the tools participating in this edition of the competition. Section 4 presents the results from the different tools. Section 5 discusses results related to tools that can produce probabilistic guarantees for falsification. Finally, Section 6 presents our reflections and conclusions.

Data Availability. The models and validation results produced by this competition are available through the shared GitLab repository at `https://gitlab.com/goranf/ARCH-COMP`, notably in the subfolders `models/FALS` and `2025/FALS`. This archive contains the results of validation and instructions to re-validate the results.

2 Benchmark

The tools participating in the competition had to consider two different procedures for generating input signals for our models, specified via an appropriate set of parameters. Section 2.1

Table 1: Minimum Simulation time and the sample step for all the benchmark models.

Model	Simulation time	Sample step
AT	33s	0.01s
AFC	51s	0.01s
NN	40s	0.01s
CC	100s	0.01s
F16	15s	0.01s
SC	35s	0.01s
PM	10s	0.01s

summarizes the rules for parameterizing the inputs for this edition of the competition. Section 2.2 presents the benchmark models and requirements.

2.1 Input Parameterization

We considered two input parameterizations: (a) arbitrary piecewise continuous input signals and (b) constrained input signals. These two parameterizations are indicated as "Instance 1" and "Instance 2".

Arbitrary piecewise continuous input signals (Instance 1). The participants can select their input parameterization, but must ensure that their inputs are piecewise continuous input signals (i.e., discontinuities are permitted). Each input signal shall assume values (for each dimension) from a given range. Participants may instruct their tools to search a subset of the entire search space, notably to achieve finite parametrization, and then to apply an interpolation scheme to synthesize the input signal.

The participants agreed that their configurations must be "reasonable" (i.e., they should be justified by the problem's specification without introducing additional knowledge about the solutions) and more general parametrizations shared across requirements and benchmark models are preferable. However, due to the diversity of benchmarks, it was decided to evaluate the proposed solutions using common sense.

Constrained input signals (Instance 2). The participants have a fixed format of the input signal. The format allows discontinuities. For example, an input signal is defined as a piecewise constant signal with k equally spaced control points. The format also defines the ranges for each input dimension, disabling interpolation at Simulink input ports so that tools don't need to up-sample their inputs.

2.2 Models and Requirements

We briefly describe our benchmark models. We also report their simulation time and sample step (Table 1) and formulae for their requirements (Table 2).

Automatic Transmission (AT) - [21].[1] A vehicle controller selects the gear (from 1 to 4) depending on the values assumed by two inputs (throttle, brake) and the current engine load, rotations per minute ω, and car speed v.

[1] Derived from a model proposed by Mathworks.

Input specification: The inputs must satisfy the following constraint $0 \leq \mathit{throttle} \leq 100$ and $0 \leq \mathit{brake} \leq 325$ (*throttle* and *brake* can be active simultaneously). Constrained input signals (instance 2) permit discontinuities at most every five time units. Requirements (specified in Table 2) are specific versions of those in [21].

Fuel Control of an Automotive Powertrain (AFC) - [23]. A controller for the air-fuel ratio in an automotive powertrain engine.

Input specification: The constrained input signal (instance 2) fixes the throttle (θ) to be piecewise constant with 10 segments over a time horizon of 50 with two modes (normal and power corresponding to feedback and feedforward control), and the engine speed ω to be constant with $900 \leq \omega < 1100$. We do not consider the unconstrained (instance 1) input specification. Faults are disabled (e.g. `fault_time` is set to a value greater than 50).

Neural-network Controller (NN) - [11].[2] An NN controller that ensures that after changes to the reference, the actual position eventually stabilizes around that value with a small error. The model has one input, a reference value *Ref* for the position. It outputs the current position of the levitating magnet *Pos*.

Input specification: The reference value *Ref* for the position must satisfy the following constraint $1 \leq \mathit{Ref} \leq 3$. The input specification for instance 1 requires discontinuities to be at least 3 time units apart, Instance 2 specifies an input signal with exactly three constant segments. The time horizon for the problem is 40.

Chasing cars (CC) - [22]. This benchmark is a set of five cars. The first car is driven by inputs (*throttle* and *brake*), and other four are driven by Hu et al.'s [22] algorithm. The output of the system is the location of five cars y_1, y_2, y_3, y_4, y_5.

Input specification: The input specification for instance 1 requires piecewise continuous signals. The instance 2 input specification constraints inputs to be piecewise constant signals with control points at every 5 seconds, i.e., 20 segments.

Aircraft Ground Collision Avoidance System (F16) - derived from [20]. The controller for Ground Collision avoidance of an F16 aircraft. The model uses 16 continuous variables and piecewise nonlinear differential equations. Autonomous maneuvers are regulated by a finite-state machine with guards on continuous variables. The system must always avoid hitting the ground during its maneuver.

Input specification: The initial conditions for roll, pitch, and yaw are in the range

$$[0.2\pi, 0.2833\pi] \times [-0.4\pi, -0.35\pi] \times [-0.375\pi, -0.125\pi].$$

This benchmark has no time-varying input. Therefore, there is no distinction between instance 1 and instance 2. The requirement is checked for a time horizon equal to 15.

Steam condenser with Recurrent Neural Network Controller (SC) [37]. A model of a steam condenser that balances the energy and the cooling water mass via a Recurrent Neural Network. The time horizon for the problem is 35 seconds.

Input specification: The input to the system can vary in the range $[3.99, 4.01]$. The instance 2 input signal should be piecewise constant with 20 evenly spaced segments.

[2] Derived from a model from Mathworks https://au.mathworks.com/help/deeplearning/ug/design-narma-l2-neural-controller-in-simulink.html.

Pacemaker (PM) [5]. A controller for a pacemaker device. The pacemaker artificially contracts the heart muscle when no natural activity is present for a given time.

Input specification: The input is the desired lower rate limit that can change within the range $[50, 90]$. The instance 2 input signal should be piecewise constant with 5 evenly spaced segments.

Synthetic Benchmark (SB) [38]. This set of benchmarks is automatically generated using `FalBenchGen`, a tool designed to address the lack of benchmarks in the falsification community. Each SB instance (denoted SB1 - SB5) is a model trained from the input and output traces that satisfy a specific STL formula. Each SB instance takes one or two input signals (ranging within $[0, 1]$), and produces either a single signal b or two signals b_1 and b_2, depending on the instance.

Input specification: The simulation time horizon for all SB instances is fixed at 24 seconds. For instance 2, all input signals must be piecewise constant with discontinuities at least 6 seconds apart. We do not consider the unconstrained (instance 1) input specification.

3 Participants

We present all participating tools (in alphabetical order) and summarize their working principle. We also provide details on how each tool was configured for the competition.

3.1 ARIsTEO

Description. ARIsTEO [31][3] is a Matlab toolbox for test case generation. It is developed on top of S-TaLiRo [3]. ARIsTEO targets a large and practically important category of CPS models, known as *compute-intensive* CPS (CI-CPS) models. CI-CPS models take hours to complete a single simulation. ARIsTEO supports CI-CPS model by embedding black-box testing into an iterative approximation-refinement loop. At the start, some sampled inputs and outputs of the model under test are used to generate a surrogate model that is faster to execute and can be subjected to black-box testing. The failure-revealing tests identified for the surrogate model are checked on the original model. If spurious, the test results are used to refine the surrogate model to be tested again. Otherwise, the test reveals a valid failure. ARIsTEO is publicly available under the General Public License (GPL).[4]

Setup. ARIsTEO provides the same interface and parameters as S-TaLiRo extended with additional configuration options. We used an ARX model (ARX-2) with order $na = 2$, $nb = 2$, and $nk = 2$[5] as a structure for the surrogate model. For models with multiple inputs and outputs, the dimensions of the matrices na, nb, and nk are changed depending on the number of inputs and outputs. We used the default configuration of S-TaLiRo for searching failure-revealing tests on the surrogate model. We considered the same parametrization of S-TaLiRo for the input signals. The original Simulink model was executed once to learn the initial surrogate model. The cut-off values for the number of simulations of the original model and the surrogate model (per trial) were set to 1500.

[3] Participants: Menghi and Formica.
[4] https://github.com/SNTSVV/ARIsTEO
[5] https://nl.mathworks.com/help/ident/ref/arx.html

Table 2: Requirement formulas for the benchmarks

Key	STL formula	Remarks/Constraints
AT1	$\square_{[0,20]} v < 120$	
AT2	$\square_{[0,10]} \omega < 4750$	
AT51	$\square_{[0,30]}((\neg g1 \wedge \circ\, g1) \rightarrow \circ\, \square_{[0,2.5]} g1)$	where $\circ\, \phi \equiv \diamond_{[0.001,0.1]}\ \phi$
AT52	$\square_{[0,30]}((\neg g2 \wedge \circ\, g2) \rightarrow \circ\, \square_{[0,2.5]} g2)$	
AT53	$\square_{[0,30]}((\neg g3 \wedge \circ\, g3) \rightarrow \circ\, \square_{[0,2.5]} g3)$	
AT54	$\square_{[0,30]}((\neg g4 \wedge \circ\, g4) \rightarrow \circ\, \square_{[0,2.5]} g4)$	
AT6a	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,4]} v < 35)$	
AT6b	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,8]} v < 50)$	
AT6c	$(\square_{[0,30]} \omega < 3000) \rightarrow (\square_{[0,20]} v < 65)$	
AT6abc	AT6a $\wedge$ AT6b $\wedge$ AT6c	conjunctive requirement
AFC27	$\square_{[11,50]}((\mathit{rise} \vee \mathit{fall}) \rightarrow (\square_{[1,5]} \lvert\mu\rvert < \beta))$	$0 \leq \theta < 61.2$ (normal mode)
AFC29	$\square_{[11,50]} \lvert\mu\rvert < \gamma$	$0 \leq \theta < 61.2$ (normal mode)
AFC33	$\square_{[11,50]} \lvert\mu\rvert < \gamma$	$61.2 \leq \theta \leq 81.2$ (power mode)
	where $\beta = 0.008$, $\gamma = 0.007$ $\mathit{rise} = (\theta < 8.8) \wedge (\diamond_{[0,0.05]}(\theta > 40.0))$ $\mathit{fall} = (\theta > 40.0) \wedge (\diamond_{[0,0.05]}(\theta < 8.8))$	
NN	$\square_{[1,37]}(\lvert Pos - Ref\rvert > \alpha + \beta\lvert Ref\rvert \rightarrow \diamond_{[0,2]} \square_{[0,1]} \neg(\alpha + \beta\lvert Ref\rvert \leq \lvert Pos - Ref\rvert))$ where $\alpha = 0.005$ and $\beta = 0.03$	
NNx	$\diamond_{[0,1]}(Pos > 3.2) \wedge \diamond_{[1,1.5]}(\square_{[0,0.5]}(1.75 < Pos < 2.25)) \wedge \square_{[2,3]}(1.825 < Pos < 2.175)$	conjunctive requiremet $1.95 \leq Ref \leq 2.05$
CC1	$\square_{[0,100]} y_5 - y_4 \leq 40$	
CC2	$\square_{[0,70]} \diamond_{[0,30]} y_5 - y_4 \geq 15$	
CC3	$\square_{[0,80]}((\square_{[0,20]} y_2 - y_1 \leq 20) \vee (\diamond_{[0,20]} y_5 - y_4 \geq 40))$	
CC4	$\square_{[0,65]} \diamond_{[0,30]} \square_{[0,5]} y_5 - y_4 \geq 8$	
CC5	$\square_{[0,72]} \diamond_{[0,8]}((\square_{[0,5]} y_2 - y_1 \geq 9) \rightarrow (\square_{[5,20]} y_5 - y_4 \geq 9))$	
CCx	$\bigwedge_{i=1..4} \square_{[0,50]}(y_{i+1} - y_i > 7.5)$	conjunctive requirement
F16	$\square_{[0,15]}\, \mathit{altitude} > 0$	
SC	$\square_{[30,35]}(87 \leq \mathit{pressure} \wedge \mathit{pressure} \leq 87.5)$	
PM	$\square_{[0,10]}(\mathit{paceCount} \leq 15) \wedge \diamond_{[0,10]}(\mathit{paceCount} \geq 8)$	
SB1	$\square_{[0,24]} b < 20$	
SB2	$\square_{[0,18]}\Big(b > 90 \vee (\diamond_{[0,6]} b < 50)\Big)$	
SB3	$(\diamond_{[6,12]} b > 10) \rightarrow (\square_{[18,24]} b > -10)$	
SB4	$\square_{[0,19]}\Big((\square_{[0,5]} b_1 \leq 20) \vee (\diamond_{[0,5]} b_2 \geq 40)\Big)$	
SB5	$\square_{[0,17]}\Bigg(\diamond_{[0,2]}\Big((\neg\square_{[0,1]} b_1 \geq 9) \vee (\square_{[1,5]} b_2 \geq 9)\Big)\Bigg)$	

3.2 ATheNA

Description. ATheNA [18][6] is a Matlab toolbox for test case generation driven by a combination of automatic and manual fitness functions. The manually-defined fitness functions are designed by the engineer and can consider the model inputs and outputs. ATheNA employs S-TaLiRo to compute the automatic fitness function from the MTL/STL specification. The model inputs are generated by an optimization algorithm minimizing the value of the ATheNA fitness. ATheNA enables the engineer to focus the exploration of the input space on particularly critical areas and to switch between different fitness functions depending on the situation.

[6] Participants: Formica and Menghi.

Setup. ATheNA has the same interface as S-TaLiRo, but requires additional information on the manual and the ATheNA fitness functions. We defined a manual fitness function for each model and requirement by reverse-engineering the model. The ATheNA fitness is the weighted average of the automatic and manual values. The weight of the two values depends on our confidence in the capabilities of the functions, leading to the identification of a fault. A brief description of the manual fitness functions and the weight used for the automatic fitness for each requirement is reported in Table 3. The manual and ATheNA fitness functions used in Instance 1 and Instance 2 are the same. The search algorithm used is Simulated Annealing, and the maximum number of iterations for the search process is 1500.

3.3 `FalCAuN`

Description. `FalCAuN` [36][7] is a toolkit for testing a Simulink model using black-box checking [34], an automated testing method based on active automata learning and model checking. In `FalCAuN`, the input and output signals given to the Simulink model under testing are discretized in time and values, and the model is deemed a black-box transition system with potentially infinite states. `FalCAuN` learns an approximation of the transition system as a Mealy machine and conducts model checking to find a counterexample. By reusing the learned Mealy machine, `FalCAuN` is designed to falsify a Simulink model against multiple specifications efficiently. `FalCAuN` is publicly available under General Public License (GPL) v3[8].

`FalCAuN` uses the *discrete-time* semantics of STL, which is essentially the same as the semantics of LTL. Because of such discretization, the control points must be fine enough to capture the timing constraints in the STL formula. For example, to capture the timing constraint $\Diamond_{[0,0.05]}$, the duration between the control points must be at most 0.05. Due to this restriction, `FalCAuN` cannot handle the STL formulas with such small timing constraints. Also, since some of the behaviors between control points are ignored in discrete-time semantics, the falsification results may deviate from the standard semantics. We remark that the current version of `FalCAuN` can handle the maximum and minimum values between control points, which prevents the above deviation if the temporal operators are not nested. The current version of `FalCAuN` only supports piecewise linear signals, and we have no results for instance 2.

Setup. For the signal discretization, we have the following parameters: The (constant) duration step of the intervals between control points, the possible values I of input signals at control points, and the thresholds of output signal values for discretization. We use the shortest duration between the control points, such that the LTL encoding of the STL formula is small enough for the back-end model checker LTSMin. The duration ranges from 0.5 to 5.0 time units. We used the constants in the given STL formulas as the thresholds of the output signal values. Table 4 summarizes the parameters we used.

3.4 FlexiFal

Description. FlexiFal [27][9] is a surrogate model-based falsification framework for CPS. The framework treats CPS as a black box and only assumes that the system to be falsified can be simulated/executed. The framework mainly contains two algorithms: NNFal [26], which utilizes a feed-forward neural network as a surrogate model of the CPS, and DTFal [27], which

[7]Participant: Waga.
[8]https://github.com/MasWag/FalCAuN
[9]Participants: Kundu, Gon, and Ray

Table 3: Manual fitness functions description and weight used by ATheNA for the benchmark requirements. The weight refers to the automatic fitness function; the manual fitness weight is equal to 1 minus the automatic fitness weight.

Benchmark	Weight	Manual Fitness Description
AT1	0.4	Maximizes the lowest throttle value within $[0, 17]$s and minimizes the highest brake value within $[0, 25]$s.
AT2	0.5	Maximizes the average throttle value within $[0, 8]$s, then minimizes the average brake value within $[0, 25]$s.
AT51	0.0	Makes the first three throttle control points get as close as possible to $\{35\%, 0\%, 50\%\}$ respectively and maximize brake within $[0, 25]$s.
AT52	0.5	Maximizes the minimum throttle value between $[0, 8]$s.
AT53	0.4	Makes the first three throttle control points get as close as possible to $\{100\%, 20\%, 0\%\}$ respectively and minimize brake within $[0, 25]$s.
AT54	0.0	Makes the first three throttle control points form an upward arc and the brake ones a downward arc.
AT6a	0.6	Makes the average throttle value within $[0, 33]$s as close as possible to 45% and minimizes the average brake value within $[0, 25]$s.
AT6b	0.4	Makes the average throttle value within $[0, 33]$s as close as possible to 45% and minimizes the average brake value within $[0, 25]$s.
AT6c	0.8	Makes the average throttle value within $[0, 33]$s as close as possible to 45% and minimizes the average brake value within $[0, 25]$s.
AT6abc	0.5	Makes the average throttle value within $[0, 33]$s as close as possible to 45% and minimizes the average brake value within $[0, 25]$s.
AFC27	0.2	Increases the two control points adjacent to the lowest one above 40 deg, then minimizes the lowest value within $[10, 50]$s.
AFC29	0.5	Minimizes the lowest throttle value within $[10, 50]$s.
AFC33	0.5	Minimizes the engine speed value.
NN	0.2	Minimizes the reference position control point at 20s.
NNx	0.5	Maximizes the lowest reference position within $[0, 20]$s.
CC1	0.5	Maximizes the lowest throttle value within $[0, 100]$s and minimizes the highest brake value within $[0, 100]$s.
CC2	0.4	Minimizes the highest throttle value within $[0, 100]$s and maximizes the lowest brake value within $[0, 100]$s.
CC3	0.8	Maximizes the lowest throttle value within $[0, 100]$s and minimizes the highest brake value within $[0, 100]$s.
CC4	0.5	Minimizes the minimum distance between cars 4 and 5 within $[0, 100]$s.
CC5	0.5	Makes the average throttle value within $[0, 33]$s as close as possible to 0.3 and maximizes the average brake value within $[0, 50]$s.
CCx	0.4	Maximizes the throttle control point at $0s$ and minimizes the throttle control point at $17s$.
F16	0.5	Maximizes the initial roll angle and minimizes the initial pitch angle.
SC	0.6	Maximizes the peak-to-peak distance of the steam flow rate within $[29.5, 35]$s.
PM	0.5	Minimizes the highest lower rate limit within $[0, 10]s$.
SB1	0.8	Maximizes the first input signal.
SB2	0.5	Makes the input at $0s$ as close as possible to 0.35 and the input at $6s$ to 0.
SB3	0.6	Makes the first input within $[0, 12)s$ as close as possible to 0.5, and maximizes the step size at $6s$ of the second input.
SB4	0.7	Maximizes the step size at $6s$ for both the first and second input.
SB5	0.5	Maximizes the input within $[0, 12)s$ and minimizes it within $[12, 18)s$.

Table 4: Configuration parameters for `FalCAuN`.

Model	Duration between control points	Possible input values
AT	2.0	throttle: 0.0, 50.0, 100.0; brake: 0.0, 325.0
CC	5.0	throttle: 0.0, 1.0; brake: 0.0, 1.0
SC	1.0	3.99, 4.00, 4.01
PM	0.5	50.0, 60.0, 70.0, 80.0, 90.0

employs a Decision Tree as a surrogate of the CPS under test. NNFal uses a feed-forward neural network as a surrogate model to leverage the adversarial attack algorithms targeted toward the robustness evaluation of neural networks. The safety property is examined in the surrogate model to find a counterexample using a deep neural network falsifier. The counterexample generated by the framework is the initial system configuration along with the piecewise constant input signal that drives the CPS to a safety-violating state. Since the surrogate model is an approximation of the CPS, the generated counterexample on the surrogate may be spurious. The last step of our framework is, therefore, validating the counterexample in the actual CPS. If the counterexample is found to be spurious, necessary constraints are added in the property specification to eliminate the spurious counterexample from the state space and search for a new counterexample for further investigation.

In DTFal, a Decision Tree is employed as a surrogate model to exploit its inherent interpretability for efficient falsification of CPS. The algorithm begins by generating a dataset to construct the Decision Tree, which serves as an approximation of the actual CPS. This dataset is generated through CPS simulations, where the initial set and a piecewise constant input signal act as inputs to the decision tree, and the robustness of the resulting trace is used as the output attribute. Our algorithm builds a Decision Tree on this dataset dynamically to find the initial configuration for a negative robustness value. Therefore, we search for falsified leaf nodes in the Decision Tree that predict negative robustness. If the falsifying leaf node is not available in the Decision Tree, our algorithm finds the nearest falsifying leaf nodes, which means the robustness value closest to zero. Then we generate the explanations from this falsified leaf node or the nearest falsifying leaf node for the safety specification. This explanation is then used to generate the counterexample. If the required counterexample is not found, additional data is generated, and the Decision Tree is retrained to repeat the process. This iterative process is repeated until the required counterexample is achieved or the execution budget is over. FlexiFal is publicly available.[10].

Setup. The current implementation of FlexiFal integrates two algorithms: NNFal and DTFal. For this competition, we utilized the DTFal algorithm to produce results for the given instances. DTFal algorithm dynamically builds a decision tree using the *DecisionTreeRegressor* class of *sklearn* library for the regression task. In addition, the default parameter settings have been used in the class to build the Decision tree. Since the output attribute of the Decision Tree is robustness of the resulting trace, we used Breach [11] to compute the robustness of a trace. To run the DTFal algorithm in FlexiFal, the `--e` must be set to DTFal. In the experiments, we set the `-- CE` option to 1, ensuring that the algorithm terminates the counterexample search process upon identifying a single counterexample. The size of the dataset (`--initial-traj`) and the number of random simulations to search for an explanation (`--ran_simu`) are chosen on a

[10] `https://gitlab.com/Atanukundu/FlexiFal`

trial-and-error basis, aiming to minimize these parameters while ensuring successful falsification. The algorithm supports only the constrained input signals (instance 2).

We target only the instance 2, so the values for FlexiFal in Table 5 are simply copies of those in Table 6.

3.5 ForeSee

Description. ForeSee [39][11] (FORmula Exploitation by Sequence trEE) tackles the *scale problem*. The scale problem can occur when the signals used in the specification have different scales (e.g., rpm and speed): namely, the contribution of a signal could be *masked* by another one when computing robustness. ForeSee introduces a new robustness definition, called *QB-Robustness*, which combines quantitative robustness and classical Boolean satisfaction. QB-Robustness does not require comparing (i.e., by minimum or maximum) robustness values of different sub-formulas, so possibly avoiding the scale problem. QB-Robustness requires the selection of a sequence of sub-formulas along the syntax tree of the specification for which to compute the quantitative robustness. Different sub-formulae sequences can be more or less effective in mitigating the scale problem.

ForeSee implements a falsification strategy based on a Monte Carlo Tree Search over the structure of the formal specification: first, by tree traversal, it identifies the sub-formulas sequence; then, on the leaves, it performs numerical hill-climbing optimization, with the aim of falsifying the selected sub-formulas. ForeSee is the spiritual successor of FalStar/MCTS from [15, 14]. It is publicly available under GNU General Public License (GPL) v3.[12]

Setup. ForeSee is implemented based on Breach [11]: It provides the same interface, can characterize the shape of input signals with several options, including piecewise constant, piecewise linear, pulse, etc. In this report, we considered piecewise constant input signals parametrized by the number of control points.

In the current implementation of ForeSee, only CMA-ES [4] is provided as the optimizer; this is due to our insight in the performances of different optimizers, in which CMA-ES outperforms other optimizers. However, involving other optimizers is not difficult for ForeSee, and will be considered in the future releases.

Since ForeSee technically relies on Monte Carlo Tree Search (MCTS), the hyperparameters in MCTS need to be properly selected. As a default setting, we use 0.2 as the scalar in the UCB1 algorithm, which balances *exploration* and *exploitation*; and we set 10 generations as the budget for the playout phase of MCTS.

3.6 FReaK

Description. FReaK [6][13] is a falsification framework for black-box models that uses an iterative refinement technique based on surrogate modeling. In particular, simulations are used to construct a surrogate model for the system dynamics using data-driven Koopman operator linearization. The reachable set of states are then computed and combined with an encoding of the signal temporal logic specification in a mixed-integer linear program (MILP). To determine the next sample, the MILP solver computes the least robust trajectory inside the reachable set of the surrogate model. The trajectory's initial state and input signal are then executed on

[11] Participants: Lyu, Zhang, and Arcaini

[12] `https://github.com/choshina/ForeSee`

[13] Participants: Hekal and Lew.

the original black-box system, where the specification is either falsified or additional simulation data is generated that we use to retrain the surrogate Koopman model and repeat the process. FReaK is publicly available[14].

Setup. The configuration parameters for FReaK depend on its main constituents. For learning the Koopman surrogate model, we use AutoKoopman [28], which automatically tunes for the associated hyperparameters. We use random Fourier features as observables with an upper bound of 20 observables, and apply grid-search for hyperparameter optimization. We use CORA [2] for reachability analysis and Gurobi[15] for solving the MILP optimization problems. Koopman linearization, reachability analysis and MILP optimization each have an associated time step size parameter determined by the chosen discretization. For consistency, we use the same time step across all three processes. Moreover, we use the number of control points of an input signal to determine the time step size for the processes. For instance, given a system with a time horizon of $100s$ and 10 control points, the associated time step size is $\Delta t = 10$. The only exception is the AFC benchmark, where a time step size of $\Delta t = 5$ was found to be too coarse for effective analysis, so we used $\Delta t = 1$ instead. We use a consistent number of control points for each model across all requirement formulas, where all signal are evenly spaced and interpolated with the `pchip` function or piecewise constant interpolation.

3.7 Ψ-TaLiRo

Description. Ψ-TaLiRo [35][16] is the python version of S-TaLiRo [3]. It is an open-source toolbox for temporal logic robustness-guided falsification of Cyber-Physical Systems (CPS). This toolbox is completely modular. It helps the generation of test cases for falsification of the system under test using a common interface for temporal logic monitors. While the toolbox provides inbuilt optimizers (DA, Uniform Random, etc.), one can also develop new optimizers. The toolbox is publicly available online under General Public License (GPL).[17] For this competition, we provide results with two different optimization algorithms:

1. **Conjunctive Bayesian Optimization - Large Scale (ConBO-LS)** accounts for the dependencies between two requirements and leverages their mutual information to achieve relatively early falsification. The approach also employs a sampler to select a subset of requirements that are more promising for falsification. Unlike the conventional approach, the algorithm takes the conjunction of all the requirements from a particular benchmark model and attempts to falsify this conjunction. It is important to note that this algorithm turns into a simple Bayesian Optimization if we do not have conjunctive requirements. However, once a requirement is falsified, the algorithm proceeds with the conjunction of the remaining unfalsified requirements.
2. **Part-X** adaptively partitions the search space to enclose the falsifying points, and can produce probabilistic guarantees on the presence of falsifying behaviors. The algorithm uses local Gaussian process estimates in order to adaptively branch and sample within the input space. The partitioning approach not only helps us identify the zero level-set of the specification robustness, but also to circumvent issues that rise due to the fact that the robustness is discontinuous. In fact, the only assumption we need on the robustness function is that it is a locally continuous function [33].

[14] https://github.com/Abdu-Hekal/FReaK
[15] https://www.gurobi.com/
[16] Participants: Khandait, Thibeault, Fainekos, and Pedrielli
[17] https://github.com/cpslab-asu/psy-taliro

Setup. In Ψ-TaLiRo, input signals are parameterized with control points and their corresponding timestamps (for interpolation), which then leads to the formation of an optimization problem with dimensionality based on the number of control points. The input signals and their corresponding timestamps are interpolated depending on the benchmark problem instance. For this competition, all signals have evenly spaced control points and are interpolated using the `pchip` function for instance 1, and a piecewise constant interpolation function for instance 2. We utilize RTAMT [32] for robustness calculation.

The repeatability package for ConBO-LS and Part-X is available online[18]. In the instances presented in this paper, ConBO-LS utilizes only a 1500-evaluation budget to falsify all the requirements from a particular benchmark model, rather than allocating 1500 evaluations for each individual requirement. In other words, the evaluation budget is allocated to a benchmark model. For example, in the Automatic Transmission benchmark in instance 1, the ConBO-LS tries to falsify the conjunction of all the 10 requirements. If any particular requirement is falsified, the algorithm continues with conjunction of the remaining requirements. In these experiments, the ConBO-LS optimizer samples 100 points from the search space and then sequentially samples points until all the requirements are falsified or the maximum budget of 1500 evaluations is reached. Finally, the Part-X optimizer, which provides probabilistic guarantees, starts with an initialization budget $n_0 = 20$, per-subregion budget for unclassified subregions with $n_{\text{BO}} = 20$, classified subregions budget $n_c = 50$, maximum budget $T = 2000$, number of Monte Carlo iterations $R = 20$, number of evaluations per iterations $M = 500$, number of cuts $B = 2$, and classification percentile $\delta_u = 0.05$. Also, we used $\delta_v = 0.001$ to identify dimensions that should not be branched. We provide the probabilistic guarantees in Table 7.

4 Evaluation & Validation

We present our experimental setup (Section 4.1) and results (Section 4.2).

4.1 Setup

The tools participating in the competition were instructed to run the falsification of each requirement 10 times, to account for the stochastic nature of most algorithms. The cut-off for the number of simulations imposed on the experiments was 1500. This value enables a more accurate comparison of the tools for difficult benchmarks. The participants provided their results. The results have been obtained on multiple platforms with varying resources and different MATLAB/Simulink versions.

The participants reported information related to each falsification trial per requirement, according to the reporting format available at https://gitlab.com/gernst/ARCH-COMP/-/blob/FALS/2021/FALS/Validation.md. The information includes:

- the benchmark (combination of model and requirement);
- the initial conditions and time-series input signal resulting from that trial;
- whether the signal is expected to falsify the requirement;
- if available, a robustness value derived from running the input through the model;
- the corresponding output signal, and further information such as time stamps or wall-clock times (optional).

[18] https://github.com/cpslab-asu/ARCH-Comp-2024-Repeatability

In the following, we will refer to this information as the "reported" result.

For each tool, we compute the falsification rate, i.e., the number of trials where a falsifying input was found, as well as the median and mean of the number of simulations required to find such input (not including the unsuccessful runs in the aggregate).

We continue the effort to validate results, which has been established in 2021. The overarching goal is to ensure that the comparison reported here is meaningful, and the approach taken accounts for several potential sources of error, both for technical reasons or because of human error. The hypothetical case of cheating participants was not regarded likely, and we emphasize upfront that no indication whatsoever for dishonest behavior was found. Rather, the goal is to establish a higher standard of quality of evaluation results, that can ultimately benefit any future work in simulation-based falsification: Just like the benchmark set established by this community gets adopted by experiments in the literature, validation of results using an independent reference checker should become standard, too. We validated the following

- the reported input signal adhere to the valid ranges of input for that particular model;
- the correctness of the reported verdict;
- the consistency of the reported robustness value and the verdict.

The results reported by the participants are presented in the following.

4.2 Results

Table 5 and Table 6 respectively report the results for instances 1 and 2 for each of participant. The tables also report the results obtained using a Uniform Random (UR) testing strategy; that is no optimization strategy is used in the search. For each instance, the tables report the falsification rate (FR), validated falsification rate (✓), mean number of simulations ($\overline{S}$), and median (rounded down) number of simulations ($\widetilde{S}$). $\overline{S}$ and $\widetilde{S}$ are computed using the successful executions, i.e., the executions where the falsification was successful. Empty cells indicate a lack of data for a particular benchmark due to lack of support or simply that the respective participants did not take the time to set up and/or run these experiments. For example, NNb, NNx, F16, and SC were not assessed for UR in instance 1.

For some of the tools, e.g., `FalCAuN` (CC4 — instance 1), the results were not confirmed by the validation platform due to differences between the configuration of the validation platform and the platform used to run the tool. For example, some results of `FalCAuN` could not be confirmed due to the use of the discrete-time semantics of STL, which was, to some degree, expected. Before evaluating STL formulas, `FalCAuN` discretizes the observed signals to interpret them as strings to apply standard automata-based techniques. However, this approach overlooks the behavior between observed points. As a result, the validation fails for STL formulas, for example, of the form $\Diamond\varphi$. For these cases, the value reported by the validated falsification rate (✓) column is lower than that reported by the falsification rate (FR) column.

For some tools, the participants found technical problems running the validation for some benchmarks, or the validation platform does not support the benchmark. These problems are reasonable since the validation tool is in early development, and some participants used it for the first time. For these cases, the participants reported the symbol "–" in the validated falsification rate (✓) column. For example, the validation platform currently does not support the validation of the Aircraft Ground Collision Avoidance System (F16) benchmark. We plan to address these limitations in the next edition of the competition.

The validator did not confirm the results of ARIsTEO and ATheNA for the AFC (ARIsTEO AFC27 - Instance 2) and NN (ARIsTEO and ATheNA NN - Instance 1, ARIsTEO and ATheNA NN, NN$_{\beta = 0.04}$, NNX - Instance 2) benchmark. In the case of the AFC benchmark, the tool

reported that some of the inputs were invalid, as the throttle pedal was equal to exactly 61.2 deg (not allowed, as shown in Table 2). Upon closer inspection, it was found that the actual test cases were always below 61.2 deg, but the value was so close that it was rounded to 61.2 deg when exporting the results in the .csv file. Therefore, both the validation tool and the participants were correct in their assessment, but the source of the error came from the numeric precision mandated by the format in which the participants had to report the results. This precision was limited to ensure the generated files had a reasonable size.
For the NN benchmark, the validation tool reported that several test cases were not producing a failure in the model's requirement as declared. However, the manual inspection of the test cases confirmed their correctness. Unfortunately, the validation tool was developed by one of the participants who did not join the competition this year, and we did not have time to perform an extensive debugging of the validation tool to identify the cause of the problem. Therefore, the table reports that ARIsTEO and ATheNA produced correct results for all of our benchmarks.

Table 5: Results for piecewise continuous input signals (instance 1). *FR*: falsification rate, ✓: validated falsification rate, $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations.

Tool:	**UR**				**ARIsTEO**				**ATheNA**				**FalCAuN**				**FlexiFal**				**ForeSee**				**FReaK**				**Ψ-TaLiRo**			
Approach:					ARX-2												DTFal												ConBO-LS			
Benchmark	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$
AT1	0	0	–	–	0	0	–	–	0	0	–	–	10		448.0	448					5	5	406.2	381	10	10	4.8	4	0	–	–	–
AT2	10	10	7.6	5	10	10	29.0	18	10	10	220.7	124	10		128.0	128	10		64	60	7	7	135.6	55	10	10	2.1	2	10	10	9.7	10
AT51	1	1	923.0	923	0	0	–	–	4	4	340.2	349					10		77.5	75	10	10	15	10.5	10	10	8.7	6	0	–	–	–
AT52	10	10	4.1	2	10	10	4.0	3	10	10	32.5	13					10		75	75	10	10	87.4	78.5	10	10	1.3	1	10	10	6.3	5
AT53	10	10	18.6	15	10	10	3.9	3	10	10	40.3	35					10		75	75	10	10	2.3	1.5	10	10	1.1	1	10	10	19.7	20
AT54	3	3	932.0	868	8	8	577.2	494	0	0	–	–					10		102.5	75	10	10	60.7	31.5	10	10	2.4	1	6	6	1008.3	1158
AT6a	10	10	74.4	41	5	5	24.4	13	10	10	200.6	191	10		501.0	501	10		228	210	10	10	56.8	39	10	10	7.4	5	10	10	187.4	82
AT6b	10	10	251.3	189	5	5	397.0	237	9	9	191.8	194	0	0	–	–	10		265	250	10	10	171.5	156.5	10	10	6.2	5	9	9	544.6	324
AT6c	10	10	185.2	86	10	10	276.5	205	10	10	268.3	139	10		449.0	449	10		255	250	10	10	107.6	73.5	10	10	5.9	5	10	10	249.0	209
AT6abc	10	10	58.8	33	10	10	276.5	205	10	10	150.7	160	10		616.0	616	10		200	200	10	10	133.7	108	10	10	6.4	5	10	10	65.6	47
NN	10	–	38.6	27	4	4	49.8	49	10	10	321.9	255									10	10	115.1	117	10	10	2.0	2	10	10	366.9	236
$NN_{\beta = 0.04}$					0	0	–	–	0	0	–	–									8	8	237.5	230	10	10	30.9	33	0	–	–	–
NNx					0	0	–	–	0	0	–	–									0	0	–	–	10	10	192.3	165	0	–	–	–
CC1	10	10	10.4	9	10	10	27.9	22	10	10	81.4	42	10		198.0	198	10	10	192	180	10	–	24.8	18	10	10	3.6	3	10	10	22.5	10
CC2	10	10	15.4	15	10	10	8.0	4	10	10	127.1	98	10		38.0	38					2	–	77.5	77.5	10	10	3.0	3	10	10	20.2	19
CC3	10	10	77.9	54	10	10	73.3	42	10	10	150.3	120	10		61.0	61	10	10	280	280	10	–	21.3	14.5	10	10	5.7	5	10	10	62.1	42
CC4	0	0	–	–	0	0	–	–	1	1	561.0	561	5		666.2	519					7	–	477.4	532	10	10	176.7	148	0	–	–	–
CC5	10	10	28.5	14	10	10	31.4	26	10	10	75.1	64					10	10	450	450	10	–	68.6	40.5	10	10	48.2	10	10	10	42.3	41
CCx	7	7	338.1	300	7	7	319.7	228	10	10	146.0	150	9		1019.3	936	10	10	1100	950	10	–	157.7	124	10	10	110.5	62	10	10	232.2	230
F16									10	–	171.9	158					10	-	388.5	375.5					10	-	1.0	1	10	–	123.0	125
SC					0	0	–	–	0	0	–	–	0	0	–	–					0	0	–	–	10	10	45.1	29	0	–	–	–
PM	8	–	571.8	443	6	6	435.5	447	10	10	162.7	152	0	0	–	–					0	0	–	–					10	10	201.0	201

Table 6: Results for piecewise continuous input signals (instance 2). *FR*: falsification rate, ✓: validated falsification rate, $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations.

Tool: Approach:	**UR**				**ARIsTEO** ARX-2				**ATheNA**				**FalCAuN**				**FlexiFal** DTFal				**ForeSee**				**FReaK**				**Ψ-TaLiRo** ConBO-LS			
Benchmark	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*FR*	✓	$\overline{S}$	$\widetilde{S}$
AT1	0	0	–	–	0	0	–	–	0	0	–	–									5	5	406.2	381	10	10	4.5	3	5	5	1131.7	1226
AT2	10	10	18.8	13	10	10	15.7	12	10	10	81.8	77					10		64.0	60	7	7	135.6	55	10	10	2.2	2	10	10	13.7	10
AT51	10	10	20.5	16	10	10	6.9	4	10	10	230.8	53					10		77.5	75	10	10	15.0	10.5	10	10	17.7	11	10	10	10.1	8
AT52	10	10	74.1	65	10	10	5.5	4	10	10	57.5	20					10		75.0	75	10	10	87.4	78.5	10	10	5.2	2	10	10	67.8	62
AT53	10	10	1.5	1	10	10	2.1	1	10	10	11.3	1					10		75.0	75	10	10	2.3	1.5	10	10	3.9	2	10	10	4.1	3
AT54	10	10	47.9	42	10	10	34.3	26	10	10	98.3	59					10		102.5	75	10	10	60.7	31.5	10	10	103.0	83	10	10	37.4	23
AT6a	10	10	156.6	138	10	10	224.5	116	10	10	177.4	155					10		228.0	210	10	10	56.8	39	10	10	24.6	15	10	10	304.5	231
AT6b	10	10	472.2	588	6	6	504.5	526	10	10	336.7	305					10		265.0	250	10	10	171.5	156.5	10	10	17.4	11	6	6	782.2	538
AT6c	10	10	326.8	176	8	8	404.6	237	10	10	193.9	165					10		255.0	250	10	10	107.6	73.5	10	10	15.5	9	9	9	472.1	369
AT6abc	10	10	149.0	125	8	8	404.6	237	10	10	198.1	184					10		200.0	200	10	10	133.7	108	10	10	13.1	8	10	10	239.2	213
AFC27	0	–	–	–	10	7	231.0	5	9	7	250.7	104					10	8	32.5	35	10	10	3	2.5	10	10	12.0	10	10	10	101.8	101
AFC29	10	–	25.1	19	10	10	2.0	2	10	10	26.8	16					10	10	200.0	200	10	10	1	1	10	10	3.1	2	10	10	9.1	7
AFC33	0	–	–	–	0	0	–	–	0	0	–	–					10	10	170.0	170	10	10	1	1					0	–	–	–
NN	10		277.2	158	10	10	96.0	89	10	10	58.0	46									10	10	115.1	117	10	10	37.2	25	10	10	75.9	83
$NN_{\beta = 0.04}$					2	2	62.5	62	3	3	991.3	940									8	8	237.5	230	10	10	534.0	472	0	–	–	–
NNx	8		457.1	380	10	10	61.9	5	10	10	239.3	196									0	0	–	–	10	10	165.3	65	10	10	143.7	135
CC1	10	10	16.4	9	10	10	6.6	6	10	10	81.4	42					10	10	192.0	180	10	–	24.8	18	10	10	3.0	2	10	10	18.0	10
CC2	10	10	12.4	13	10	10	6.8	3	10	10	127.1	98									2	–	77.5	77.5	10	10	3.3	3	10	10	14.8	9
CC3	10	10	19.6	21	10	10	20.4	15	10	10	150.3	120					10	10	280.0	280	10	–	21.3	14.5	10	10	2.6	2	10	10	11.4	10
CC4	0	0	–	–	0	0	–	–	1	1	561.0	561									7	–	477.4	532	10	10	1349.9	1201	1	1	1387.3	1500
CC5	10	10	37.4	22	10	10	16.7	15	10	10	75.1	64					10	10	450.0	450	10	–	68.6	40.5	10	10	47.5	34	10	10	32.2	36
CCx	6	6	396.7	284	10	10	570.6	482	10	10	146.0	150					10	10	1100.0	950	10	–	157.7	124	7	7	1723.6	938	10	10	244.2	225
SC					0	0	–	–	0	0	–	–									0	0	–	–					0	–	–	–
PM	6	–	575.8	617	9	9	553.3	620	10	10	162.7	152									0	0	–	–					10	10	201.0	201
SB1					0	0	–	–	4	4	864.8	766					3	3	7000.0	6500	0	0	–	–								
SB2					0	0	–	–	0	0	–	–									0	0	–	–								
SB3					0	0	–	–	10	10	212.9	210					10	10	1325.0	1250	5	–	562.6	593								
SB4					3	3	716.3	819	4	4	567.2	521					10	10	4250.0	3500	6	–	101.2	88								
SB5					0	0	–	–	10	10	210.3	232					9	9	4000.0	4000	2	–	101	101								

5 Probabilistic Guarantees

As done in 2022, we are still assessing tools that can provide probabilistic guarantees for falsifying the system under test to understand if we can provide any conclusion about the system under test and falsifying it. This information becomes even more critical when no falsification is found. Providing probabilistic guarantees can help assess the system's safety, while also providing the quality of test samples generated.

Part-X [33] was the only tool that provided results on probabilistic guarantees: the lower and upper confidence bounds of normalized falsification volume at 95% confidence. The Part-X algorithm is part of the Ψ-TaLiRo tool, and is discussed in section 3.7. The results are shown in Table 7 for both instances. We refrain from an in-depth analysis of these results.

Table 7: Results for piecewise continuous input signals (instance 1) and constrained input signals (instance 2). *FR*: falsification rate, ✓: validated falsification rate, $\overline{S}$: mean number of simulations, $\widetilde{S}$: median (rounded down) number of simulations, *LCB*: Lower Confidence Bound at 95% confidence, *UCB*: Upper Confidence Bound at 95% confidence *R*: Simulation time ratio (%). Bold entries indicate that some results could not be validated.

Tool	**Ψ-TaLiRo**							**Ψ-TaLiRo**						
Approach	Part-X							Part-X						
Instance	1							2						
Property	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*LCB*	*UCB*	*R*	*FR*	✓	$\overline{S}$	$\widetilde{S}$	*LCB*	*UCB*	*R*
AT1	10	10	34.9	28.5	0.00E+00	7.03E-04	70.7	10	10	30.5	25.5	0.00E+00	5.58E-04	85.7
AT2	10	10	6.7	5.5	9.45E-02	1.80E-01	52.8	10	10	6.5	5.0	1.16E-01	2.77E-01	50.6
AT51	0	0	–	–	0.00E+00	0.00E+00	93.9	10	10	13.3	11.5	2.22E-01	5.86E-01	64.3
AT52	10	10	5.6	2.0	1.81E-01	9.02E-01	62.5	10	10	66.5	53.5	0.00E+00	0.00E+00	93.5
AT53	10	10	15.7	15.5	2.45E-02	4.26E-01	59.7	10	10	2.2	2.0	8.38E-01	1.00E+00	57.0
AT54	3	3	862.6	–	0.00E+00	3.60E-05	91.0	10	10	85.0	65.0	0.00E+00	7.68E-02	76.2
AT6a	10	10	134.3	51.5	1.18E-01	2.47E-01	58.2	10	10	153.7	72.0	5.75E-02	1.94E-01	53.1
AT6b	10	10	212.2	150.0	9.45E-02	2.88E-01	57.8	10	10	307.9	111.5	3.40E-02	1.97E-01	56.4
AT6c	10	10	200.5	138.0	9.94E-02	2.86E-01	58.1	10	10	334.4	249.5	4.34E-02	1.98E-01	59.3
AT6abc	10	10	126.1	50.0	1.02E-01	2.67E-01	68.7	10	10	106.9	67.5	5.72E-02	2.06E-01	69.4
CC1	10	10	19.0	16.5	2.71E-01	8.31E-01	69.2	10	10	17.6	21.0	2.83E-01	8.86E-01	68.7
CC2	10	10	23.9	12.0	4.82E-01	1.00E+00	68.6	10	10	17.8	12.0	2.27E-01	1.00E+00	66.3
CC3	10	**9**	23.1	24.0	1.28E-01	4.58E-01	69.9	10	10	13.5	12.0	1.18E-01	1.00E+00	69.5
CC4	0	0	–	–	0.00E+00	0.00E+00	95.3	0	0	–	–	0.00E+00	0.00E+00	94.5
CC5	10	10	45.8	29.0	3.83E-02	7.10E-01	79.4	10	10	29.9	22.5	2.09E-01	5.90E-01	73.8
CCx	9	9	681.9	703.0	0.00E+00	0.00E+00	96.0	10	10	607.1	156.0	0.00E+00	0.00E+00	96.2
NN	10	10	15.2	16.0	4.84E-01	8.80E-01	83.5	10	10	145.8	89.5	0.00E+00	1.36E-01	87.3
NNx	–	–	–	–	–	–	–	10	10	190.7	40.0	0.00E+00	1.20E-02	66.4
SC	0	–	–	–	0.00E+00	0.00E+00	78.9	0	0	–	–	0.00E+00	2.70E-05	45.5
F16	0	–	–	–	0.00E+00	0.00E+00	39.2	–	–	–	–	–	–	–
AFC27	–	–	–	–	–	–	–	10	0	34.3	27.0	5.90E-01	7.27E-01	89.4
AFC29	–	–	–	–	–	–	–	10	0	12.1	11.0	2.31E-01	5.36E-01	87.9
AFC33	–	–	–	–	–	–	–	0	0	–	–	0.00E+00	0.00E+00	96.1
PM	10	10	23.5	22.0	6.75E-3	7.13E-3	80.9	8	8	253.6	26.5	0.00E+00	3.44E-3	82.7

6 Conclusion and Outlook

Seven tools participated in the 2025 edition: ARIsTEO [31], ATheNA [18], FalCAuN [36], FlexiFal [27], ForeSee [39], FReaK [6], and Ψ-TaLiRo [35]. Each of these tools uses a completely different approach to generate test cases and falsify the requirements under analysis, so a direct comparison between them is challenging. However, the results reported in Tables 5 and 6 can provide a starting point for this comparison. We also noticed that tools based on approximation-refinement techniques [31] driven by surrogate models increased over the years: ARIsTEO [31] (participating since 2020), FlexiFal [27] (participating since 2023), and FReaK [6] (participating since 2024).

Based on the available data, no tool has a definitive advantage over all the others, and all the available solutions have their pros and cons. Therefore, we cannot define a "winner" for this year's competition. An engineer should carefully evaluate the advantages and disadvantages of each participant and choose the approach that best matches their individual needs. Note that not all participants were able to support all the requirement or all the set of assumptions on the input signals. Furthermore, some tools require tuning of their algorithms for each experiment, which can be very time-consuming. For these reasons, we allowed groups to participate even with a partial evaluation.

Tanmay Khandait was crucial in validating the results of all participants. He provided technical support to all the groups and improved the tool by extending the validation also to the Pacemaker (PM) model, which was not supported last year, and all the new Synthetic Benchmark models (SB1–SB5). Deyun Lyu proposed five new models for this year competition, based on Recurrent Neural Networks and LSTM cells. These new models consider requirements with different combinations of temporal and boolean operators. Remarkably, none of the participants were able to produce a failure-revealing test case for the SB2 model, which provides an interesting challenge for future editions of the competition.

There are several points in our agenda that we would like to improve in the next year. We would like to reintroduce the ratio between the simulation time and the total computational time, after formalizing its definition. We plan to extend the validation tool to the Aircraft Ground Collision Avoidance System (F16) benchmark, which is currently the only model not supported. We also plan to look into all the discrepancies between the participating tools and the validation platform and make sure that the validation tool does not have any bugs (and, in case, fix them). We also found an increasing interest in using Python models, so we plan to revise the rules of the competition to also consider Python benchmarks. Finally, we would like to encourage the participants to provide probabilistic guarantees with their results.

Acknowledgments The experiments for ARIsTEO and ATheNA were in part enabled by the support provided by Compute Ontario (computeontario.ca) and the Digital Research Alliance of Canada (alliancecan.ca). C. Menghi is supported by the European Union – Next Generation EU. "Sustainable Mobility Center (Centro Nazionale per la Mobilitá Sostenibile - CNMS)", M4C2 – Investment 1.4, Project Code CN_00000023, and by projects SERICS (PE00000014) under the NRRP MUR program, and GLACIATION (101070141). P. Arcaini is supported by ASPIRE grant No. JPMJAP2301, JST. The ASU team (Ψ-TaLiRo) was partially supported by DARPA FA8750-20-C-0507, NSF CMMI 2046588, NSF CNS 2000792, and NSF CMMI 1829238. M. Waga is partially supported by JST PRESTO Grant Number JPMJPR22CA, JST BOOST Grant Number JPMJBY24H8, JSPS KAKENHI Grant Number JP22K17873, and JST CREST Grant Number JPMJCR2012, Japan. Z. Zhang is supported by JST BOOST Grant No. JPMJBY24D7 and JSPS Grant No. JP25K21179.

References

[1] Alessandro Abate, Matthias Althoff, Lei Bu, Gidon Ernst, Goran Frehse, Luca Geretti, Taylor T. Johnson, Claudio Menghi, Stefan Mitsch, Stefan Schupp, and Sadegh Soudjani. The arch-comp friendly verification competition for continuous and hybrid systems. In *TOOLympics Challenge 2023*, pages 1–37, Cham, 2025. Springer Nature Switzerland.

[2] M. Althoff. An introduction to CORA 2015. In *Proc. of the International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 120–151, 2015.

[3] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 254–257. Springer, 2011.

[4] Anne Auger and Nikolaus Hansen. A restart CMA evolution strategy with increasing population size. In *IEEE Congress on Evolutionary Computation, CEC 2005*, pages 1769–1776, 2005.

[5] Mostafa Ayesh, Namya Mehan, Ethan Dhanraj, Abdul El-Rahwan, Simon Emil Opalka, Tony Fan, Akil Hamilton, Akshay Mathews Jacob, Rahul Anthony Sundarrajan, Bryan Widjaja, and Claudio Menghi. Two simulink models with requirements for a simple controller of a pacemaker device. In *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, EPiC Series in Computing, pages 18–25. EasyChair, 2022.

[6] Stanley Bak, Sergiy Bogomolov, Abdelrahman Hekal, Niklas Kochdumper, Ethan Lew, Andrew Mata, and Amir Rahmati. Falsification using reachability of surrogate Koopman models. In *Proceedings of the 27th ACM International Conference on Hybrid Systems: Computation and Control*, pages 1–13, 2024.

[7] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: a survey on theory, tools and applications. In *Lectures on Runtime Verification*, pages 135–175. Springer, 2018.

[8] Anthony Corso, Robert J Moss, Mark Koren, Ritchie Lee, and Mykel J Kochenderfer. A survey of algorithms for black-box safety validation. *arXiv preprint arXiv:2005.02979*, 2020.

[9] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, and Georgios Fainekos. ARCH-COMP17 category report: Preliminary results on the falsification benchmarks. In *ARCH17. International Workshop on Applied Verification of Continuous and Hybrid Systems*, EPiC Series in Computing, pages 170–174. EasyChair, 2017.

[10] Adel Dokhanchi, Shakiba Yaghoubi, Bardh Hoxha, Georgios Fainekos, Gidon Ernst, Zhenya Zhang, Paolo Arcaini, Ichiro Hasuo, and Sean Sedwards. ARCH-COMP18 category report: Results on the falsification benchmarks. In *ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems*, EPiC Series in Computing, pages 104–109. EasyChair, 2018.

[11] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pages 167–170. Springer, 2010.

[12] Johan Liden Eddeland, Alexandre Donze, Sajed Miremadi, and Knut Akesson. Industrial temporal logic specifications for falsification of cyber-physical systems. In *ARCH@CPSIoTWeek*, 2020.

[13] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Aniruddh Chandratre, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Khouloud Gaaloul, Jun Inoue, Tanmay Khandait, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Masaki Waga, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP 2021 category report: Falsification with validation of results. In *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, EPiC Series in Computing, pages 133–152. EasyChair, 2021.

[14] Gidon Ernst, Paolo Arcaini, Ismail Bennani, Alexandre Donzé, Georgios Fainekos, Goran Frehse, Logan Mathesen, Claudio Menghi, Giulia Pedrielli, Marc Pouzet, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP 2020 category report: Falsification. In *ARCH20.*

International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), EPiC Series in Computing, pages 140–152. EasyChair, 2020.

[15] Gidon Ernst, Paolo Arcaini, Alexandre Donzé, Georgios Fainekos, Logan Mathesen, Giulia Pedrielli, Shakiba Yaghoubi, Yoriyuki Yamagata, and Zhenya Zhang. ARCH-COMP 2019 category report: Falsification. In *ARCH19. International Workshop on Applied Verification of Continuous and Hybrid Systems*, EPiC Series in Computing, pages 129–140. EasyChair, 2019.

[16] Gidon Ernst, Paolo Arcaini, Georgios Fainekos, Federico Formica, Jun Inoue, Tanmay Khandait, Mohammad Mahdi Mahboob, Claudio Menghi, Giulia Pedrielli, Masaki Waga, Yoriyuki Yamagata, and Zhenya Zhang. Arch-comp 2022 category report: Falsification with ubounded resources. In *International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH22)*, EPiC Series in Computing, pages 204–221. EasyChair, 2022.

[17] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In Klaus Havelund, Manuel Núñez, Grigore Roşu, and Burkhart Wolff, editors, *Formal Approaches to Software Testing and Runtime Verification*, LNCS, pages 178–192. Springer, 2006.

[18] Federico Formica, Tony Fan, and Claudio Menghi. Search-based software testing driven by automatically generated and manually defined fitness functions. *ACM Trans. Softw. Eng. Methodol.*, 33(2), December 2023.

[19] Martin Fränzle and Michael R Hansen. A robust interpretation of duration calculus. In *International Colloquium on Theoretical Aspects of Computing*, pages 257–271. Springer, 2005.

[20] Peter Heidlauf, Alexander Collins, Michael Bolender, and Stanley Bak. Verification challenges in F-16 ground collision avoidance and other automated maneuvers. In *ARCH18. International Workshop on Applied Verification of Continuous and Hybrid Systems*, EPiC Series in Computing, pages 208–217. EasyChair, 2018.

[21] Bardh Hoxha, Houssam Abbas, and Georgios Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *ARCH14-15. International Workshop on Applied veRification for Continuous and Hybrid Systems*, EPiC Series in Computing, pages 25–30. EasyChair, 2015.

[22] Jianghai Hu, John Lygeros, and Shankar Sastry. Towards a theory of stochastic hybrid systems. In *International Workshop on Hybrid Systems: Computation and Control*, pages 160–173. Springer, 2000.

[23] Xiaoqing Jin, Jyotirmoy V. Deshmukh, James Kapinski, Koichi Ueda, and Ken Butts. Powertrain control verification benchmark. In *International Conference on Hybrid Systems: Computation and Control*, pages 253–262. ACM, 2014.

[24] Tanmay Khandait, Federico Formica, Paolo Arcaini, Surdeep Chotaliya, Georgios Fainekos, Abdelrahman Hekal, Atanu Kundu, Ethan Lew, Michele Loreti, Claudio Menghi, Laura Nenzi, Giulia Pedrielli, Jarkko Peltomäki, Ivan Porres, Rajarshi Ray, Valentin Soloviev, Ennio Visconti, Masaki Waga, and Zhenya Zhang. Arch-comp 2024 category report: Falsification. In *11th International Workshop on Applied Verification of Continuous and Hybrid Systems. ARCH24*, volume 103, pages 122–144. EasyChair, 2024.

[25] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[26] Atanu Kundu, Sauvik Gon, and Rajarshi Ray. Data-driven falsification of cyber-physical systems. In *Proceedings of the 17th Innovations in Software Engineering Conference*, pages 1–5, 2024.

[27] Atanu Kundu, Sauvik Gon, and Rajarshi Ray. Data-driven falsification of cyber-physical systems. *arXiv preprint arXiv:2505.03863*, 2025.

[28] E. Lew and et al. AutoKoopman: A toolbox for automated system identification via Koopman operator linearization. In *Proc. of the International Symposium on Automated Technology for Verification and Analysis*, pages 237–250, 2023.

[29] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In Yassine Lakhnech and Sergio Yovine, editors, *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[30] Claudio Menghi, Paolo Arcaini, Walstan Baptista, Gidon Ernst, Georgios Fainekos, Federico Formica, Sauvik Gon, Tanmay Khandait, Atanu Kundu, Giulia Pedrielli, Jarkko Peltomäki, Ivan Porres, Rajarshi Ray, Masaki Waga, and Zhenya Zhang. Arch-comp 2023 category report: Falsification. In *10th International Workshop on Applied Verification of Continuous and Hybrid Systems. ARCH23*, volume 96, pages 151–169, 2023.

[31] Claudio Menghi, Shiva Nejati, Lionel Briand, and Yago Isasi Parache. Approximation-refinement testing of compute-intensive cyber-physical models: An approach based on system identification. In *International Conference on Software Engineering (ICSE)*. IEEE / ACM, 2020.

[32] Dejan Ničković and Tomoya Yamaguchi. RTAMT: Online Robustness Monitors from STL. In *Automated Technology for Verification and Analysis: International Symposium (ATVA)*, page 564–571. Springer-Verlag, 2020.

[33] Giulia Pedrielli, Tanmay Khandait, Yumeng Cao, Quinn Thibeault, Hao Huang, Mauricio Castillo-Effen, and Georgios Fainekos. Part-X: A family of stochastic algorithms for search-based test generation with probabilistic guarantees. *IEEE Transactions on Automation Science and Engineering*, pages 1–22, 2023.

[34] Doron Peled, Moshe Y Vardi, and Mihalis Yannakakis. Black box checking. In *Formal Methods for Protocol Engineering and Distributed Systems: FORTE XII/PSTV XIX'99 IFIP TC6 WG6. International Conference on Formal Description Techniques for Distributed Systems and Communication Protocols (FORTE XII) and Protocol Specification, Testing and Verification (PSTV XIX)*, pages 225–240. Springer, 1999.

[35] Quinn Thibeault, Jacob Anderson, Aniruddh Chandratre, Giulia Pedrielli, and Georgios Fainekos. PSY-TaLiRo: A Python Toolbox for Search-Based Test Generation for Cyber-Physical Systems. In *Formal Methods for Industrial Critical Systems*, pages 223–231. Springer, 2021.

[36] Masaki Waga. Falsification of cyber-physical systems with robustness-guided black-box checking. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 11:1–11:13. ACM, 2020.

[37] Shakiba Yaghoubi and Georgios Fainekos. Gray-box adversarial testing for control systems with machine learning components. In *International Conference on Hybrid Systems: Computation and Control (HSCC)*, 2019.

[38] Yipei Yan, Deyun Lyu, Zhenya Zhang, Paolo Arcaini, and Jianjun Zhao. Automated generation of benchmarks for falsification of STL specifications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1–14, 2025.

[39] Zhenya Zhang, Deyun Lyu, Paolo Arcaini, Lei Ma, Ichiro Hasuo, and Jianjun Zhao. Effective Hybrid System Falsification Using Monte Carlo Tree Search Guided by QB-Robustness. In *Computer Aided Verification*, pages 595–618. Springer, 2021.